**mysql学习笔记**

1. **mysql安装**

材料准备：mysql-5.7.16-linux-glibc2.5-x86\_64.tar.gz、centOS

**1.1解压**

|  |
| --- |
| tar -zxvf mysql-5.7.16-linux-glibc2.5-x86\_64.tar.gz |

1.2移动到/usr/local/下

|  |
| --- |
| mv mysql-5.7.16-linux-glibc2.5-x86\_64 /usr/local/mysql |

**1.3新建mysql用户**

|  |
| --- |
| useradd -d /home/mysql -m mysql |

**1.4新建一个mysqldata文件夹**

|  |
| --- |
| mkdir -vp /home/mysql/mysqldata |

**1.5将/home/mysql/mysqldata文件夹拥有者改为mysql**

|  |
| --- |
| chown -R mysql:mysql /home/mysql |

**1.6初始化数据库**

|  |
| --- |
| /usr/local/mysql/bin/mysqld --initialize  --user=mysql  --basedir=/usr/local/mysql/ --datadir=/home/mysql/mysqldata |
| 注意：(1)会生成一个临时密码，一定要记住，后面会用到  (2)如果使用/usr/local/mysql/bin/mysqld --initialize-insecure 来初始化，则不会生成临时密码。 |

**1.7开启ssl连接**

|  |
| --- |
| /usr/local/mysql/bin/mysql\_ssl\_rsa\_setup --user=mysql --basedir=/usr/local/mysql --datadir=/home/mysql/mysqldata |

**1.8 创建my.cnf文件**

|  |
| --- |
| vim /etc/my.cnf |
| [client]  default-character-set=utf8  [mysqld]  character-set-server=utf8  datadir=/home/mysql/mysqldata  basedir=/usr/local/mysql  socket=/tmp/mysql.sock  ##不区分大小写  lower\_case\_table\_names=1  skip-ssl  ##默认端口是3306,端口不是3306需要制定  port=3306  max\_connections = 1000  [mysqld\_safe]  #log-error=/home/mysql/log/mysqld.log<br>pid-file=/home/mysql/pid/mariadb.pid |

**1.8配置mysqld服务**

|  |
| --- |
| cp /usr/local/mysql/support-files/mysql.server /etc/init.d/mysqld |
| vim /etc/init.d/mysqld  ##修改basedir= 自己的路径 修改datadir= 自己的路径 |

**1.9开启mysqld服务**

|  |
| --- |
| service mysqld start |

**1.10 配置mysql的环境变量**

|  |
| --- |
| vim /etc/profile |
| #在最后加上  ####set mysql  export PATH=/usr/local/mysql/bin:${PATH} |
| source /etc/profile |

**1.11进入mysql修改密码**

|  |
| --- |
| mysql -u root -p |
| ##修改密码  set password =password('qnhl1911'); |
| ##如果外网可以访问，需要给外网的账号授权：  grant all privileges on \*.\* to root@'%' identified by 'qnhl1911';  flush privileges; |

**1.12开放端口**

|  |
| --- |
| ##此时在外网使用客户端还不可以访问，可以考虑先关闭防火墙试试，如果关闭防火墙后##可以访问，应该是端口权限问题。可以给端口授权  firewall-cmd --zone=public --add-port=3306/tcp --permanent |
| ##然后需要重新启动防火墙才可以生效  systemctl stop firewalld  systemctl start firewalld |

**1.13设置开机自启动**

|  |
| --- |
| chmod 755 /etc/init.d/mysql  chkconfig --add mysql  chkconfig --level 345 mysql on |

1. **mysql多实例安装**

**2.1背景**

 MySQL数据库的集中化运维，可以通过在一台服务器上，部署运行多个MySQL服务进程，通过不同的socket监听不同的服务端口来提供各自的服务。各个实例之间是相互独立的，每个实例的datadir, port, socket, pid都是不同的。

**2.2多实例特点**

 有效利用服务器资源，当单个服务器资源有剩余时，可以充分利用剩余的资源提供更多的服务。  
  资源互相抢占问题，当某个服务实例服务并发很高时或者开启慢查询时，会消耗更多的内存、CPU、磁盘IO资源，导致服务器上的其他实例提供服务的质量下降。

|  |
| --- |
| IMG_256 |

**2.3 mysql多实例安装方式**

mysql的多实例有两种方式可以实现，两种方式各有利弊。  
第一种是使用多个配置文件启动不同的进程来实现多实例，这种方式的优势逻辑简单，配置简单，缺点是管理起来不太方便。  
第二种是通过官方自带的mysqld\_multi使用单独的配置文件来实现多实例，这种方式定制每个实例的配置不太方面，优点是管理起来很方便，集中管理。

下面就分别来实战这两种多实例的安装和管理

**2.4多配置文件安装**

前提条件：执行以下操作需要mysql已经安装，如果没有按照，按照之前的方式安装即可

**2.4.1停mysqld服务**

|  |
| --- |
| pkill mysqld  ps -ef|grep mysql  rm -f /etc/init.d/mysqld |

**2.4.2创建目录并授权**

|  |
| --- |
| mkdir -vp /data/mysql\_data{5..6}/data |
| chown -R mysql.mysql /data/ |

**2.4.3在/data/mysql\_data{5..6}/创建配置文件my.cnf**

|  |
| --- |
| [client]  port=3310  socket=/data/mysql\_data5/mysql.sock  [mysql]  no-auto-rehash  [mysqld]  server-id=5  user=mysql  port=3310  socket=/data/mysql\_data5/mysql.sock  basedir=/usr/local/mysql  datadir=/data/mysql\_data5/data  open\_files\_limit=10240  back\_log=600  max\_connections=3000  max\_connect\_errors=6000  table\_open\_cache=614  external-locking=FALSE  max\_allowed\_packet=32M  thread\_cache\_size=300  query\_cache\_size=64M  query\_cache\_limit=4M  default-storage-engine=InnoDB  thread\_stack=192K  transaction\_isolation=READ-COMMITTED  tmp\_table\_size=256M  max\_heap\_table\_size=256M  long\_query\_time=4  slow\_query\_log\_file=/data/mysql\_data5/slow-log.log  slow\_query\_log  log-bin=/data/mysql\_data5/ray-bin.log  binlog\_cache\_size=4M  max\_binlog\_cache\_size=8M  max\_binlog\_size=512M  expire\_logs\_days=7  key\_buffer\_size=256MB  read\_buffer\_size=1M  read\_rnd\_buffer\_size=16M  bulk\_insert\_buffer\_size=64M  skip-name-resolve  lower\_case\_table\_names=1  innodb\_buffer\_pool\_size=512M  innodb\_thread\_concurrency=8  innodb\_flush\_log\_at\_trx\_commit=2  innodb\_log\_buffer\_size=16M  innodb\_log\_file\_size=128M  innodb\_log\_files\_in\_group=3  innodb\_max\_dirty\_pages\_pct=90  innodb\_lock\_wait\_timeout=120  innodb\_file\_per\_table=on  [mysqldump]  quick  max\_allowed\_packet=32M  [mysqld\_safe]  log-error=/data/mysql\_data5/mysql\_ray.err  pid-file=/data/mysql\_data5/ray.pid |
| [client]  port=3311  socket=/data/mysql\_data6/mysql.sock  [mysql]  no-auto-rehash  [mysqld]  server-id=6  user=mysql  port=3311  socket=/data/mysql\_data6/mysql.sock  basedir=/usr/local/mysql  datadir=/data/mysql\_data6/data  open\_files\_limit=10240  back\_log=600  max\_connections=3000  max\_connect\_errors=6000  table\_open\_cache=614  external-locking=FALSE  max\_allowed\_packet=32M  thread\_cache\_size=300  query\_cache\_size=64M  query\_cache\_limit=4M  default-storage-engine=InnoDB  thread\_stack=192K  transaction\_isolation=READ-COMMITTED  tmp\_table\_size=256M  max\_heap\_table\_size=256M  long\_query\_time=4  slow\_query\_log\_file=/data/mysql\_data6/slow-log.log  slow\_query\_log  #log-bin=/data/mysql\_data6/ray-bin.log  binlog\_cache\_size=4M  max\_binlog\_cache\_size=8M  max\_binlog\_size=512M  expire\_logs\_days=7  key\_buffer\_size=256MB  read\_buffer\_size=1M  read\_rnd\_buffer\_size=16M  bulk\_insert\_buffer\_size=64M  skip-name-resolve  lower\_case\_table\_names=1  innodb\_buffer\_pool\_size=512M  innodb\_thread\_concurrency=8  innodb\_flush\_log\_at\_trx\_commit=2  innodb\_log\_buffer\_size=16M  innodb\_log\_file\_size=128M  innodb\_log\_files\_in\_group=3  innodb\_max\_dirty\_pages\_pct=90  innodb\_lock\_wait\_timeout=120  innodb\_file\_per\_table=on  [mysqldump]  quick  max\_allowed\_packet=32M  [mysqld\_safe]  log-error=/data/mysql\_data6/mysql\_ray.err  pid-file=/data/mysql\_data6/ray.pid |

**2.4.4初始化各个实例**

|  |
| --- |
| /usr/local/mysql/bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data5 |
| /usr/local/mysql/bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data6 |
| **注意：需要记住初始化的密码，后面登录需要用到** |

**2.4.5开启各实例的ssl连接**

|  |
| --- |
| /usr/local/mysql/bin/mysql\_ssl\_rsa\_setup --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data5 |
| /usr/local/mysql/bin/mysql\_ssl\_rsa\_setup --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data16 |

**2.4.6多实例服务启动和关闭**

|  |
| --- |
| # 多实例启动文件的启动mysql服务实质：  /usr/local/mysql/bin/mysqld\_safe --defaults-file=/data/mysql\_data5/my.cnf 2>&1 > /dev/null &  /usr/local/mysql/bin/mysqld\_safe --defaults-file=/data/mysql\_data6/my.cnf 2>&1 > /dev/null & |
| # 多实例启动文件的停止mysql服务实质：  /usr/local/mysql/bin/mysqladmin -u root -p -S /data/mysql\_data5/mysql.sock shutdown  /usr/local/mysql/bin/mysqladmin -u root -p -S /data/mysql\_data6/mysql.sock shutdown |

**2.4.7登录重新修改密码**

|  |
| --- |
| Mysql -u root -p -S /data/mysql\_data5/mysql.sock |
| Mysql -u root -p -S /data/mysql\_data6/mysql.sock |

**后面的操作和数据安装中一样**

**2.5 mysqld\_multi多实例安装**

**2.5.1 创建目录**

|  |
| --- |
| mkdir -vp /data/mysql\_data{1..4} |

**2.5.2 修改 MySQL 多实例数据目录与 数据导入/导出专放目录的所属用户与所属组**

|  |
| --- |
| chown mysql.mysql -R /usr/local/mysql/mysql-files /data/mysql\_data{1..4} |

**2.5.3配置 MySQL 配置文件 /etc/my.cnf**

|  |
| --- |
| [mysqld\_multi]  mysqld    = /usr/local/mysql/bin/mysqld   mysqladmin = /usr/local/mysql/bin/mysqladmin log        = /tmp/mysql\_multi.log     [mysqld1]  # 设置数据目录　[多实例中一定要不同]  datadir = /data/mysql\_data1 # 设置sock存放文件名　[多实例中一定要不同]  socket = /tmp/mysql.sock1  # 设置监听开放端口　[多实例中一定要不同]  port = 3306  # 设置运行用户  user = mysql  # 关闭监控  performance\_schema = off  # 设置innodb 缓存大小  innodb\_buffer\_pool\_size = 32M  # 设置监听IP地址  bind\_address = 0.0.0.0  # 关闭DNS 反向解析  skip-name-resolve = 0     [mysqld2]  datadir = /data/mysql\_data2 socket = /tmp/mysql.sock2  port = 3307  user = mysql  performance\_schema = off  innodb\_buffer\_pool\_size = 32M  bind\_address = 0.0.0.0  skip-name-resolve = 0     [mysqld3]  datadir = /data/mysql\_data3 socket = /tmp/mysql.sock3  port = 3308  user = mysql  performance\_schema = off  innodb\_buffer\_pool\_size = 32M  bind\_address = 0.0.0.0  skip-name-resolve = 0     [mysqld4]  datadir = /data/mysql\_data4 socket = /tmp/mysql.sock4  port = 3309  user = mysql  performance\_schema = off  innodb\_buffer\_pool\_size = 32M  bind\_address = 0.0.0.0  skip-name-resolve = 0 |

**2.5.4 初始化各个实例 [ 初始化完成后会自带随机密码在输出日志中 ]**

|  |
| --- |
| /usr/local/mysql/bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data1  /usr/local/mysql/bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data2  /usr/local/mysql/bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data3  /usr/local/mysql/bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data4 |

**2.5.5各实例开启 SSL 连接**

|  |
| --- |
| /usr/local/mysql/bin/mysql\_ssl\_rsa\_setup --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data1   /usr/local/mysql/bin/mysql\_ssl\_rsa\_setup --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data2   /usr/local/mysql/bin/mysql\_ssl\_rsa\_setup --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data3   /usr/local/mysql/bin/mysql\_ssl\_rsa\_setup --user=mysql --basedir=/usr/local/mysql --datadir=/data/mysql\_data4 |

**2.5.6复制多实例脚本到服务管理目录下 [ /etc/init.d/ ]**

|  |
| --- |
| cp /usr/local/mysql/support-files/mysqld\_multi.server /etc/init.d/mysqld\_multi |

**2.5.7添加脚本执行权限**

|  |
| --- |
| chmod +x /etc/init.d/mysqld\_multi |

**2.5.8添加进service服务管理**

|  |
| --- |
| chkconfig --add mysqld\_multi （centos） |
| sudo update-rc.d ServiceName defaults (ubuntu) |

**2.5.9启动多实例**

|  |
| --- |
| /etc/init.d/mysqld\_multi start 注意：在启动多实例之前最好先 source /etc/profile |
| /etc/init.d/mysqld\_multi report |

**2.5.10 修改各个密码**

|  |
| --- |
| 与配置文件一致，最好每个实例的密码设置一样 |

1. **mysql登录与关闭**

**Mysql启动：**

|  |
| --- |
|  |

**Mysql关闭**

|  |
| --- |
| **Mysqladmin -u root -p shutdown** |

**Mysql登录**

|  |
| --- |
| **Mysql -u root -p 回车** |
| **多实例登录：**  **Mysql -u root -p -S /tmp/mysql.sock （本机操作）** |
| **Mysql -u root -p (多实例远程)** |

1. **mysql小窍门**

**4.1 mysql prompt的用法详解**

--prompt=name 参数对于做运维的人来说很重要。我们登入mysql数据库后，mysql的提示符只是一个很简单内容 mysql> ,没有其他任何信息。通过--prompt=name可以自定义提示信息，通过配置显示登入的主机地址，登陆用户名，当前时间，当前数据库schema等待。个人强烈要求加上 登入主机名，登陆用户名，当前库 schema这三项加入提示内容。

|  |
| --- |
| mysql -u root -p -S /tmp/mysql.sock1 --prompt="\\u@\\h:\\d \\r:\\m:\\s>" |
|  |

这种方式只是临时生效。

2、也可以把参数加到my.cnf里面

|  |
| --- |
| vim /etc/my.cnf  [mysql] prompt="[\\u@\\h](/home/wupeng/文档\\x/u@//h" \t "/home/wupeng/文档\\x/_blank) :[\\d](/home/wupeng/文档\\x/d" \t "/home/wupeng/文档\\x/_blank)[\\r:\\m:\\s](r://m://s" \t "/home/wupeng/文档\\x/_blank)>" |

**4.2不重启修改全局参数的方案**

Mysql的全局参数在my.cnf中配置，再修改了my.cnf中的全局参数的时候，需要重启数据库才能生效，而在mysql中修改全局参数，虽然立即生效，但是在重新启动mysql的数据库之后依然失效。

问：如何才能在不重启的状态修改全局参数，在重启后生效呢？

答： 现在mysql中修改当前状态的全局变量，使之立即生效，然后修改my.cnf中的对应参数，使之在以后重启后依然生效。

演示实例：

|  |
| --- |
| 1、现在max\_connections=3000  show variables like '%connect%';  +-----------------------------------------------+--------------------+  | Variable\_name | Value |  +-----------------------------------------------+--------------------+  | character\_set\_connection | utf8mb4 |  | collation\_connection | utf8mb4\_general\_ci |  | connect\_timeout | 10 |  | disconnect\_on\_expired\_password | ON |  | init\_connect | |  | max\_connect\_errors | 6000 |  | max\_connections | 3000 |  | max\_user\_connections | 0 |  | performance\_schema\_session\_connect\_attrs\_size | 512 |  +-----------------------------------------------+--------------------+ |
| 2、修改max\_connections，使之本次生效  MySQL [(none)]> set global max\_connections=4000;  Query OK, 0 rows affected (0.00 sec) |
| 3、在my.cnf中修改max\_connections=4000,使之在下次重启的时候生效。  vim /data/mysql\_data3306/my.cnf |

**备注：一般开关类的参数，使用set global修改无效。**

4.3生产环境常用的命令

|  |
| --- |
| show status; ----查看当前会话的数据库信息 |
| show global status; --查看整个数据库的运行状态信息，很重要，要分析并做好监控。 |
| show processlist; --查看正在执行的sql语句，不全 |
| show full processlist; --查看正在执行的sql语句，全 |
| Set global max\_connections=4000; --不重启调整数据库的参数，直接生效。重启失效。 |
| show svariables; --查看数据库的参数信息 |

1. **创建用户及授权**

**5.1 方法一**

|  |
| --- |
| **CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';**  **GRANT ALL ON db1.\* TO 'jeffrey'@'localhost';** |

**5.2 方法二**

|  |
| --- |
| **grant all privileges on \*.\* to guest@'%' identified by 'pms#$123%';** |

**5.3 mysql的权限列表**

**All privileges包含：**

|  |
| --- |
| **SELECT，**  **INSERT,**  **UPDATE,**  **DELETE,**  **CREATE,**  **DROP,**  **RELOAD,**  **SHUTDOWN,**  **PROCESS,**  **FILE,**  **REFERENCES,**  **INDEX,**  **ALTER,**  **SHOW DATABASES,**  **SUPER,**  **CREATE TEMPORARY TABLES,**  **LOCK TABLES,**  **EXECUTE,**  **REPLICATION SLAVE,**  **REPLICATION CLIENT,**  **CREATE VIEW,**  **SHOW VIEW,**  **CREATE ROUTINE,**  **ALTER ROUTINE,**  **CREATE USER,**  **EVENT,**  **TRIGGER,**  **CREATE TABLESPACE** |

**备注：**

**(1)一般来说授权采用最小原则，尽量不要使用grant all privileges ..,应该尽量使用**

**Grant select,insert,update,delete on \*.\* to guest@@'%' identified by 'pms#$123%';**

1. **如何知道all privileges包含哪些权限？可以先给一个用户授予说有权限，然后revoke select权限，然后再使用 show grants for user@’’的用法。**

**5.4 取消一个用户的授权**

|  |
| --- |
| **REVOKE INSERT ON \*.\* FROM 'jeffrey'@'localhost';** |

1. **mysql主从复制**

**5.1概述**

Mysql内建的复制功能是构建大型，高性能应用程序的基础。将Mysql的数据分布到多个系统上去，这种分布的机制，是通过将Mysql的某一台主机的数据复制到其它主机（slaves）上，并重新执行一遍来实现的。复制过程中一个服务器充当主服务器，而一个或多个其它服务器充当从服务器。主服务器将更新写入二进制日志文件，并维护文件的一个索引以跟踪日志循环。这些日志可以记录发送到从服务器的更新。当一个从服务器连接主服务器时，它通知主服务器从服务器在日志中读取的最后一次成功更新的位置。从服务器接收从那时起发生的任何更新，然后封锁并等待主服务器通知新的更新。

请注意当你进行复制时，所有对复制中的表的更新必须在主服务器上进行。否则，你必须要小心，以避免用户对主服务器上的表进行的更新与对从服务器上的表所进行的更新之间的冲突。

**5.2主从复制搭建**

此处采用的是mysql多实例配置来搭建主从复制

主服务器：127.0.0.1 3306

从服务器：127.0.0.1 3307

**5.2.1关闭数据库服务**

**5.2.2主服务器master配置**

|  |
| --- |
| sudo vim /data/mysql\_data3306/my.cnf |
| log-bin=/data/mysql\_data3306/ray-bin.log  server-id=1 |

5.2.3主服务器slave配置

|  |
| --- |
| sudo vim /data/mysql\_data3307/my.cnf |
| log-bin=/data/mysql\_data3307/ray-bin.log  server-id=2  #库名.表名 如果不设置则表示所有的都同步。  replicate-do-table=wishrp.sku\_map |

**5.2.4重启两台mysql服务**

|  |
| --- |
| 关闭：  mysqladmin -u root -p password -S /data/mysql\_data3306/mysql.sock shutdown  mysqladmin -u root -p password -S /data/mysql\_data3307/mysql.sock shutdown |
| 开启：  mysqld\_safe --defaults-file=/data/mysql\_data3306/my.cnf 2>&1 > /dev/null &  mysqld\_safe --defaults-file=/data/mysql\_data3307/my.cnf 2>&1 > /dev/null & |

**5.2.5在主服务器上建立帐户并授权slave:**

|  |
| --- |
| 登录：mysql -u root -p -S /data/mysql\_data3306/mysql.sock |
| mysql>GRANT REPLICATION SLAVE ON \*.\* to 'mysync'@'%' identified by 'q123456'; |
| 一般不用root帐号，表示所有客户端都可能连，只要帐号，密码正确，此处可用具体客户端IP代替，如192.168.145.226，加强安全。 |

5.2.6**登录主服务器的mysql，查询master的状态**

|  |
| --- |
| mysql>show master status; |
| +----------------+----------+--------------+------------------+-------------------+  | File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |  +----------------+----------+--------------+------------------+-------------------+  | ray-bin.000002 | 439 | | | |  +----------------+----------+--------------+------------------+-------------------+  1 row in set (0.00 sec) |
| 注：执行完此步骤后不要再操作主服务器MYSQL，防止主服务器状态值变化 |

5.2.7**配置从服务器Slave：**

|  |
| --- |
| mysql> change master to master\_host='127.0.0.1',master\_user='mysync',master\_password='q123456',master\_log\_file='ray-bin.000002',master\_log\_pos=439; |

5.2.7slave开启复制并检查检查状态

|  |
| --- |
| mysql> start slave; |
| mysql> show slave status\G;  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Slave\_IO\_State: Waiting for master to send event  Master\_Host: 127.0.0.1 //主服务器地址  Master\_User: mysync /授权帐户名，尽量避免使用root  Master\_Port: 3306  Connect\_Retry: 60  Master\_Log\_File: ray-bin.000002  Read\_Master\_Log\_Pos: 439  Relay\_Log\_File: wupeng-relay-bin.000002  Relay\_Log\_Pos: 318  Relay\_Master\_Log\_File: ray-bin.000002  Slave\_IO\_Running: Yes //此状态必须YES  Slave\_SQL\_Running: Yes  //此状态必须YES  Replicate\_Do\_DB:  Replicate\_Ignore\_DB:  Replicate\_Do\_Table:  Replicate\_Ignore\_Table:  Replicate\_Wild\_Do\_Table:  Replicate\_Wild\_Ignore\_Table:  Last\_Errno: 0  Last\_Error:  Skip\_Counter: 0  Exec\_Master\_Log\_Pos: 439  Relay\_Log\_Space: 526  Until\_Condition: None  Until\_Log\_File:  Until\_Log\_Pos: 0  Master\_SSL\_Allowed: No  Master\_SSL\_CA\_File:  Master\_SSL\_CA\_Path:  Master\_SSL\_Cert:  Master\_SSL\_Cipher:  Master\_SSL\_Key:  Seconds\_Behind\_Master: 0  Master\_SSL\_Verify\_Server\_Cert: No  Last\_IO\_Errno: 0  Last\_IO\_Error:  Last\_SQL\_Errno: 0  Last\_SQL\_Error:  Replicate\_Ignore\_Server\_Ids:  Master\_Server\_Id: 1  Master\_UUID: fbc5f1d0-ef79-11e7-a697-54e1ad1c2c14  Master\_Info\_File: /data/mysql\_data3307/data/master.info  SQL\_Delay: 0  SQL\_Remaining\_Delay: NULL  Slave\_SQL\_Running\_State: Slave has read all relay log; waiting for more updates  Master\_Retry\_Count: 86400  Master\_Bind:  Last\_IO\_Error\_Timestamp:  Last\_SQL\_Error\_Timestamp:  Master\_SSL\_Crl:  Master\_SSL\_Crlpath:  Retrieved\_Gtid\_Set:  Executed\_Gtid\_Set:  Auto\_Position: 0  Replicate\_Rewrite\_DB:  Channel\_Name:  Master\_TLS\_Version: |
| Slave\_IO及Slave\_SQL进程必须正常运行，即YES状态，否则都是错误的状态(如：其中一个NO均属错误)。 以上操作过程，主从服务器配置完成。 |

**5.2.8 测试主从**

**主从同步如果确保数据不丢：**

1. **主库挂掉，切换到从库，使用主库的binlog日志补全;**
2. **双写;**
3. **程序保留一分钟数据**
4. **谷歌的办同步插件**
5. **数据备份与恢复**

**读写分离：**

**中大型公司：程序**

**测试环境：mysql-proxy,amoeba**

**门户网站：分布式dbproxy**

**6.1mysqldump备份**

|  |
| --- |
| **备份一个数据库的所有的数据和表结构** |
| mysqldump -u root -p -h 192.168.1.98 -P 13308 --databases buslf > e:mysql\_bak\buslf.sql |
| 还原： |
| mysql -u root -p -h 192.168.1.98 -P 13308<e:mysql\_bak\tsintergy\_wrf\_history02.sql |
|  |

参数简介：

-u 帐号

-p 密码

-h ip地址，如果不添加此参数，默认是本机

-P 端口号，不添加默认是3306

-t 只备份数据

|  |
| --- |
| mysqldump -uroot -p -B -d --compact -S /data/mysql\_data3306/mysql.sock test |
| INSERT INTO `s\_nation` VALUES (1,'11','2111'),(2,'hanzu','2'),(3,'hanzu','1'),(7,'111','11');  INSERT INTO `s\_school` VALUES (3,'name3'),(4,'土家族'),(5,'汉族'); |

-d 只备份表结构

|  |
| --- |
| mysqldump -uroot -p -B -d --compact -S /data/mysql\_data3306/mysql.sock test |
| CREATE DATABASE /\*!32312 IF NOT EXISTS\*/ `test` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;  USE `test`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!40101 SET character\_set\_client = utf8 \*/;  CREATE TABLE `s\_nation` (  `id` int(11) NOT NULL,  `nation\_name` varchar(45) DEFAULT NULL,  `code` varchar(45) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!40101 SET character\_set\_client = utf8 \*/;  CREATE TABLE `s\_school` (  `id` int(11) NOT NULL,  `s\_name` varchar(45) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/; |

-B 添加B参数的时候，导出的备份文件中会有创建数据库的语句，再还原的时候不需要额外的建立数据库

|  |
| --- |
| CREATE DATABASE /\*!32312 IF NOT EXISTS\*/ `test` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;  USE `test`; |

--compact 此参数使地输出的内容更少，适合调试使用，不建议在生产环境使用

|gzip 可以使用管道命令压缩数据

|  |
| --- |
| sudo mysqldump -uroot -p -B -S /data/mysql\_data3306/mysql.sock test|gzip >/home/wupeng/dumps/test20180106\_B.sql.gz |
| 由此可见，gzip压缩可节省3倍空间  -rw-r--r-- 1 wupeng sudo 2775 1月 6 11:45 test20180106\_B.sql  -rw-r--r-- 1 wupeng sudo 897 1月 6 12:04 test20180106\_B.sql.gz |

--备份多个库 直接在后面加多个数据库

|  |
| --- |
| sudo mysqldump -uroot -p -B -S /data/mysql\_data3306/mysql.sock test test1|gzip >/home/wupeng/dumps/test20180106\_B.sql.gz |

-A 备份数据库所有的东西

-F 切割binlog。在musqldump的时候，会执行flush logs操作。重新生成一个binlog

--master-data=1 不刷新binlog.生成的备份数据中会出现出现的binlog的位置点。

|  |
| --- |
| mysqldump -uroot -p -t --compact --master-data=1 -S /data/mysql\_data3306/mysql.sock test |
| CHANGE MASTER TO MASTER\_LOG\_FILE='ray-bin.000002', MASTER\_LOG\_POS=154;  INSERT INTO `s\_nation` VALUES (1,'11','2111'),(2,'hanzu','2'),(3,'hanzu','1'),(7,'111','11');  INSERT INTO `s\_school` VALUES (3,'name3'),(4,'土家族'),(5,'汉族'); |

-x 锁表

-l 所表for read

生产中备份：

Myisam:

Mysqldump -u root -p -A -B --master-data=1 -x|gzip

innoDb

Mysqldump -u root -p -A -B --master-data=1 --single-transaction|gzip

**6.2 source恢复**

source恢复需要进入mysql中执行。

|  |
| --- |
| 1、先查看有哪些数据库  mysql> show databases;  +--------------------+  | Database |  +--------------------+  | information\_schema |  | mysql |  | performance\_schema |  | sys |  | test |  | test1 |  +--------------------+  6 rows in set (0.01 sec) |
| 1. 删除数据库test   mysql> drop database test;  Query OK, 2 rows affected (0.03 sec) |
| 3、执行source根据备份的文件来获取  mysql> source /home/wupeng/dumps/test20180106\_B.sql |

6.4使用mysql恢复

|  |
| --- |
| 1. 如果在mysqldump在导出数据的时候加的有-B参数，此时在恢复的时候不需要指定数据库，因为在备份文件中已经存在创建数据库的语句，此时使用   mysql -u root -p -S /data/mysql\_data3306/mysql.sock </home/wupeng/dumps/test20180106\_B.sql |
| 1. 如果在mysqldump在导出数据的时候没有加-B参数，此时在恢复的时候需要指定数据库，因为在备份文件中已经存在创建数据库的语句，此时使用   mysql -u root -p test -S /data/mysql\_data3306/mysql.sock </home/wupeng/dumps/test20180106\_B.sql |

**6.5mysqlbinlog**

众所周知，binlog日志对于mysql数据库来说是十分重要的。在数据丢失的紧急情下，我们往往会想到用binlog日志功能进行数据恢复（定时全备份+binlog日志恢复增量数据部分），化险为夷！

废话不多说，下面是梳理的binlog日志操作解说：

6.2.1初步了解binlog

MySQL的二进制日志binlog可以说是MySQL最重要的日志，它记录了所有的DDL和DML语句（除了数据查询语句select），以事件形式记录，还包含语句所执行的消耗的时间，MySQL的二进制日志是事务安全型的。

DDL

----Data Definition Language 数据库定义语言

主要的命令有CREATE、ALTER、DROP等，DDL主要是用在定义或改变表（TABLE）的结构，数据类型，表之间的链接和约束等初始化工作上，他们大多在建立表时使用。

DML

----Data Manipulation Language 数据操纵语言

主要的命令是SELECT、UPDATE、INSERT、DELETE，就象它的名字一样，这4条命令是用来对数据库里的数据进行操作的语言

mysqlbinlog常见的选项有以下几个：

--start-datetime：从二进制日志中读取指定等于时间戳或者晚于本地计算机的时间

--stop-datetime：从二进制日志中读取指定小于时间戳或者等于本地计算机的时间 取值和上述一样

--start-position：从二进制日志中读取指定position 事件位置作为开始。

--stop-position：从二进制日志中读取指定position 事件位置作为事件截至

一般来说开启binlog日志大概会有1%的性能损耗。

binlog日志有两个最重要的使用场景:

1）MySQL主从复制：MySQL Replication在Master端开启binlog，Master把它的二进制日志传递给slaves来达到master-slave数据一致的目的。

2）自然就是数据恢复了，通过使用mysqlbinlog工具来使恢复数据。

binlog日志包括两类文件：

1）二进制日志索引文件（文件名后缀为.index）用于记录所有的二进制文件

2）二进制日志文件（文件名后缀为.00000\*）记录数据库所有的DDL和DML(除了数据查询语句select)语句事件。

6.2.2开启binlog日志：

1）编辑打开mysql配置文件my.cnf,在[mysqld] 区块添加log-bin=/data/mysql\_data3306/ray-bin.log确认是打开状态(mysql-bin 是日志的基本名或前缀名)；

2）重启mysqld服务使配置生效

注意：

Binlog\_format有三种模式：

statement:在binlog中存储的是sql语句

|  |
| --- |
| wupeng@wupeng:/data/mysql\_data3306$sudo mysqlbinlog --no-defaults ./ray-bin.000006 |
| /\*!50530 SET @@SESSION.PSEUDO\_SLAVE\_MODE=1\*/;  /\*!40019 SET @@session.max\_insert\_delayed\_threads=0\*/;  /\*!50003 SET @OLD\_COMPLETION\_TYPE=@@COMPLETION\_TYPE,COMPLETION\_TYPE=0\*/;  DELIMITER /\*!\*/;  # at 4  #180104 17:45:42 server id 1 end\_log\_pos 123 CRC32 0x4bd57c93 Start: binlog v 4, server v 5.7.16-log created 180104 17:45:42  # Warning: this binlog is either in use or was not closed properly.  BINLOG '  xvdNWg8BAAAAdwAAAHsAAAABAAQANS43LjE2LWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  AAAAAAAAAAAAAAAAAAAAAAAAEzgNAAgAEgAEBAQEEgAAXwAEGggAAAAICAgCAAAACgoKKioAEjQA  AZN81Us=  '/\*!\*/;  # at 123  #180104 17:45:42 server id 1 end\_log\_pos 154 CRC32 0x007bc542 Ignorable  # Ignorable event type 35 (MySQL Previous\_gtids)  # at 154  #180104 17:46:10 server id 1 end\_log\_pos 219 CRC32 0xddeffc55 Ignorable  # Ignorable event type 34 (MySQL Anonymous\_Gtid)  # at 219  #180104 17:46:10 server id 1 end\_log\_pos 294 CRC32 0xc92cc6d5 Query thread\_id=3 exec\_time=0 error\_code=0  SET TIMESTAMP=1515059170/\*!\*/;  SET @@session.pseudo\_thread\_id=3/\*!\*/;  SET @@session.foreign\_key\_checks=1, @@session.sql\_auto\_is\_null=0, @@session.unique\_checks=1, @@session.autocommit=1/\*!\*/;  SET @@session.sql\_mode=1436549152/\*!\*/;  SET @@session.auto\_increment\_increment=1, @@session.auto\_increment\_offset=1/\*!\*/;  /\*!\C utf8 \*//\*!\*/;  SET @@session.character\_set\_client=33,@@session.collation\_connection=33,@@session.collation\_server=33/\*!\*/;  SET @@session.lc\_time\_names=0/\*!\*/;  SET @@session.collation\_database=DEFAULT/\*!\*/;  BEGIN  /\*!\*/;  # at 294  #180104 17:46:10 server id 1 end\_log\_pos 408 CRC32 0x031bc97e Query thread\_id=3 exec\_time=0 error\_code=0  SET TIMESTAMP=1515059170/\*!\*/;  DELETE FROM `test`.`s\_nation` WHERE `id`='4'  /\*!\*/;  # at 408  #180104 17:46:10 server id 1 end\_log\_pos 563 CRC32 0x7d035b27 Query thread\_id=3 exec\_time=0 error\_code=0  SET TIMESTAMP=1515059170/\*!\*/;  INSERT INTO `test`.`s\_nation` (`id`, `nation\_name`, `code`) VALUES ('7', '111', '11')  /\*!\*/;  # at 563  #180104 17:46:10 server id 1 end\_log\_pos 710 CRC32 0x8b86ebf3 Query thread\_id=3 exec\_time=0 error\_code=0  SET TIMESTAMP=1515059170/\*!\*/;  UPDATE `test`.`s\_nation` SET `nation\_name`='11', `code`='2111' WHERE `id`='1'  /\*!\*/;  # at 710  #180104 17:46:10 server id 1 end\_log\_pos 741 CRC32 0x6037c675 Xid = 19  COMMIT/\*!\*/;  # at 741  #180104 18:00:22 server id 1 end\_log\_pos 806 CRC32 0x7958ce26 Ignorable  # Ignorable event type 34 (MySQL Anonymous\_Gtid)  # at 806  #180104 18:00:22 server id 1 end\_log\_pos 882 CRC32 0x24635565 Query thread\_id=3 exec\_time=0 error\_code=0  SET TIMESTAMP=1515060022/\*!\*/;  BEGIN  /\*!\*/;  # at 882  #180104 18:00:22 server id 1 end\_log\_pos 1036 CRC32 0x24700e50 Query thread\_id=3 exec\_time=0 error\_code=0  SET TIMESTAMP=1515060022/\*!\*/;  INSERT INTO `test1`.`new\_table` (`idnew\_table`, `new\_tablecol1`) VALUES ('3', '44')  /\*!\*/;  # at 1036  #180104 18:00:22 server id 1 end\_log\_pos 1190 CRC32 0xaa7cf1ab Query thread\_id=3 exec\_time=0 error\_code=0  SET TIMESTAMP=1515060022/\*!\*/;  INSERT INTO `test1`.`new\_table` (`idnew\_table`, `new\_tablecol1`) VALUES ('4', '55')  /\*!\*/;  # at 1190  #180104 18:00:22 server id 1 end\_log\_pos 1221 CRC32 0xd6b92472 Xid = 80  COMMIT/\*!\*/;  DELIMITER ;  # End of log file  ROLLBACK /\* added by mysqlbinlog \*/;  /\*!50003 SET COMPLETION\_TYPE=@OLD\_COMPLETION\_TYPE\*/;  /\*!50530 SET @@SESSION.PSEUDO\_SLAVE\_MODE=0\*/; |
| 优点：不需要记录每一行的变化，减少了binlog日志量，节约了IO，提高性能。  缺点：由于记录的只是执行语句，为了这些语句能在slave上正确运行，因此还必须记录每条语句在执行的时候的一些相关信息，以保证所有语句能在slave得到和在master端执行时候相同 的结果。另外mysql 的复制,像一些特定函数功能，slave可与master上要保持一致会有很多相关问题。  ps：相比row能节约多少性能与日志量，这个取决于应用的SQL情况，正常同一条记录修改或者插入row格式所产生的日志量还小于Statement产生的日志量，但是考虑到如果带条件的update操作，以及整表删除，alter表等操作，ROW格式会产生大量日志，因此在考虑是否使用ROW格式日志时应该跟据应用的实际情况，其所产生的日志量会增加多少，以及带来的IO性能问题。 |

Row:

|  |
| --- |
| 5.1.5版本的MySQL才开始支持row level的复制,它不记录sql语句上下文相关信息，仅保存哪条记录被修改。  优点： binlog中可以不记录执行的sql语句的上下文相关的信息，仅需要记录那一条记录被修改成什么了。所以rowlevel的日志内容会非常清楚的记录下每一行数据修改的细节。而且不会出现某些特定情况下的存储过程，或function，以及trigger的调用和触发无法被正确复制的问题.  缺点:所有的执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，这样可能会产生大量的日志内容。  ps:新版本的MySQL中对row level模式也被做了优化，并不是所有的修改都会以row level来记录，像遇到表结构变更的时候就会以statement模式来记录，如果sql语句确实就是update或者delete等修改数据的语句，那么还是会记录所有行的变更。 |
| 在row格式下，DML是加密存储，好在mysqlbinlog提供参数-v 反解析查看，指令如下：  sudo mysqlbinlog --no-defaults ./ray-bin.000002 -v |

mixed

|  |
| --- |
| 从5.1.8版本开始，MySQL提供了Mixed格式，实际上就是Statement与Row的结合。  在Mixed模式下，一般的语句修改使用statment格式保存binlog，如一些函数，statement无法完成主从复制的操作，则采用row格式保存binlog，MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在Statement和Row之间选择一种。 |

6.3常用的binlog日志操作命令  
1）查看所有binlog日志列表

|  |
| --- |
| mysql> show master logs; +------------------+-----------+ | Log\_name | File\_size | +------------------+-----------+ | mysql-bin.000001 | 149 | | mysql-bin.000002 | 4102 | +------------------+-----------+ 2 rows in set (0.00 sec) |

1. 查看master状态，即最后(最新)一个binlog日志的编号名称，及其最后一个操作事件pos结束点(Position)值

|  |
| --- |
| mysql> show master status; +------------------+----------+--------------+------------------+ | File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | +------------------+----------+--------------+------------------+ | mysql-bin.000002 | 4102 | | | +------------------+----------+--------------+------------------+ 1 row in set (0.00 sec) |

1. flush刷新log日志，自此刻开始产生一个新编号的binlog日志文件

|  |
| --- |
| mysql> flush logs;  Query OK, 0 rows affected (0.13 sec)  mysql> show master logs;  +------------------+-----------+ | Log\_name | File\_size | +------------------+-----------+ | mysql-bin.000001 | 149 | | mysql-bin.000002 | 4145 | | mysql-bin.000003 | 106 | +------------------+-----------+ 3 rows in set (0.00 sec) |

注意：  
每当mysqld服务重启时，会自动执行此命令，刷新binlog日志；在mysqldump备份数据时加 -F 选项也会刷新binlog日志；

1. 重置(清空)所有binlog日志

|  |
| --- |
| mysql> reset master; Query OK, 0 rows affected (0.12 sec)  mysql> show master logs;  +------------------+-----------+ | Log\_name | File\_size | +------------------+-----------+ | mysql-bin.000001 | 106 | +------------------+-----------+ 1 row in set (0.00 sec) |

**6.4查看binlog日志内容，常用有两种方式：**1）使用mysqlbinlog自带查看命令法：  
注意：  
-->binlog是二进制文件，普通文件查看器cat、more、vim等都无法打开，必须使用自带的mysqlbinlog命令查看  
-->binlog日志与数据库文件在同目录中  
-->在MySQL5.5以下版本使用mysqlbinlog命令时如果报错，就加上 “--no-defaults”选项

使用mysqlbinlog命令查看binlog日志内容，下面截取其中的一个片段分析：  
[root@vm-002 mysql]# mysqlbinlog mysql-bin.000002  
..............  
# at 624  
#160925 21:29:53 server id 1 end\_log\_pos 796 Query thread\_id=3 exec\_time=0 error\_code=0  
SET TIMESTAMP=1474810193/\*!\*/;  
insert into member(`name`,`sex`,`age`,`classid`) values('wangshibo','m',27,'cls1'),('guohuihui','w',27,'cls2')        #执行的sql语句  
/\*!\*/;  
# at 796  
#160925 21:29:53 server id 1 end\_log\_pos 823 Xid = 17                  #执行的时间  
.............

解释：  
server id 1 ： 数据库主机的服务号；  
end\_log\_pos 796： sql结束时的pos节点  
thread\_id=11： 线程号

6.6.查看mysql进程

|  |
| --- |
| 1. 查看目前mysql有哪些连接，使用 show processlist;命令   **sudo mysql -u root -pqinghua123@ -S /data/mysql\_data3306/mysql.sock -e 'show processlist;'**  +----+------+-----------+------+---------+------+----------+------------------+  | Id | User | Host | db | Command | Time | State | Info |  +----+------+-----------+------+---------+------+----------+------------------+  | 22 | root | localhost | NULL | Query | 0 | starting | show processlist |  +----+------+-----------+------+---------+------+----------+------------------+ |
| 1. 查看mysql连接的所有信息，使用show full processlist;   **sudo mysql -u root -pqinghua123@ -S /data/mysql\_data3306/mysql.sock -e 'show full processlist;'**  +----+------+-----------+------+---------+------+----------+-----------------------+  | Id | User | Host | db | Command | Time | State | Info |  +----+------+-----------+------+---------+------+----------+-----------------------+  | 23 | root | localhost | NULL | Query | 0 | starting | show full processlist |  +----+------+-----------+------+---------+------+----------+-----------------------+ |

# **七、MySQL利用frm和idb文件进行数据恢复**

7.1 MYISAM数据引擎

MySQL中如果用的是MYISAM数据引擎，那么数据很好恢复，只要将相应.frm, .MYD, .MYI文件拷贝过去即可

试验：

7.1.1在3306中创建一个引擎为MyISAM的表student

|  |
| --- |
| create table student(  id int(10) unsigned not null auto\_increment,  name varchar(10) character set utf8,  age int(10),  primary key(id)  ) engine=MyISAM |
|  |

7.1.2查看test数据库目录下的student文件

|  |
| --- |
| root@wupeng:/data/mysql\_data3307/data# cd ../../mysql\_data3306/data/test/  root@wupeng:/data/mysql\_data3306/data/test# ls -l  总用量 128  -rw-r----- 1 mysql mysql 61 1月 7 12:57 db.opt  -rw-r----- 1 mysql mysql 8614 1月 7 12:57 new\_table.frm  -rw-r----- 1 mysql mysql 98304 1月 7 12:58 new\_table.ibd  -rw-r----- 1 mysql mysql 8614 1月 9 21:32 student.frm  -rw-r----- 1 mysql mysql 0 1月 9 21:32 student.MYD  -rw-r----- 1 mysql mysql 1024 1月 9 21:32 student.MYI |
|  |

7.1.3 将student.frm,student.MYD,student.MYI移动到3307的数据库下面

|  |
| --- |
| root@wupeng:/data/mysql\_data3306/data/test# cp student.\* ../../../mysql\_data3307/data/hkll/ |

7.1.4 将/data/mysql\_data3307/data授予mysql,确保mysql有读写权限

|  |
| --- |
| chown -R mysql:mysql /data/mysql\_data3307/data |

7.1.6 打开3307,查询数据验证

|  |
| --- |
|  |

**7.2 InnoDB数据引擎**

如果是innodb的话，如果开启innodb\_file\_per\_table＝1,则每一个数据表都是一个单独的文件，比如User表，则会建立User.frm和User.ibd.

那么直接拷贝这两个文件到新的MySQL数据目录下可以吗，一般来说是不行的，那么什么时候可以呢？只有在你的ibd文件的tablespace id和ibdata1文件中的元信息的tablespace id一致才可以。

那么怎么办呢？innodb恢复不了就是因为table space id不匹配导致的。所以我们可以这样做 (以test数据库为例)：

数据库：3306

数据库：3307

要求：将3306的test和test1数据库恢复到3307中

**7.2.1关闭3307**

|  |
| --- |
| 关闭3307数据库  sudo /data/mysql\_data3307/mysqld.sh stop |
| 查看3307的端口,确认数据库是否关闭  sudo netstat -lntup|grep 3307 |

**7.2.2备份3307的ibdata1和数据库文件信息**

|  |
| --- |
| Sudo mv /data/mysql\_data3307/data/ibdata1 ibdata1\_bak |

**7.2.3 将3306的ibdata1，test，test1复制到3307的/data/mysql\_data3307/data/下面：**

|  |
| --- |
| root@wupeng:/data/mysql\_data3306/data# cp -r ibdata1 test test1 ../../mysql\_data3307/data/ |

**7.2.4将6.2.3复制过来的文件授权给mysql管理，这样是确保文件有读写权限**

|  |
| --- |
| wupeng@wupeng:/data/mysql\_data3307$ sudo chown -R mysql:mysql data |

**7.2.5 重新启动3307服务**

|  |
| --- |
| wupeng@wupeng:/data/mysql\_data3307$ sudo ./mysqld.sh start |

**7.2.6进入3307查看数据是否可以查看**

|  |
| --- |
|  |

**7.2.7使用mysqldump将数据从3307备份下来**

**7.2.8 将3307的数据库还原。**

**八、innoDB 只剩下ibd,frm恢复（已知建表语句）**

问题描述：现在已知basic\_ctrl\_net的建表语句，以及basic\_ctrl\_net.ibd,basic\_ctrl\_net.frm文件，如何恢复数据。

实验机器：3307

1、在3307的hklll数据库中建立表basic\_ctrl\_net

|  |
| --- |
| Use hkll;  CREATE TABLE `basic\_ctrl\_net` (  `Net\_Id` varchar(32) NOT NULL,  `Net\_Name` varchar(50) DEFAULT NULL,  `Parent\_Id` varchar(32) DEFAULT NULL,  PRIMARY KEY (`Net\_Id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8; |

2、在3307数据库上清理basic\_ctrl\_net的表空间

|  |
| --- |
| MySQL [hkll]> alter table basic\_ctrl\_net discard tablespace;  Query OK, 0 rows affected (0.01 sec) |

3、将basic\_ctrl\_net.ibd文件复制到/data/mysql\_data3307/data/hkll/下

|  |
| --- |
| root@wupeng:/home/wupeng/Downloads# cp basic\_ctrl\_net.ibd /data/mysql\_data3307/data/hkll/ |

4、重新导入basic\_ctrl\_net的表空间

|  |
| --- |
| MySQL [hkll]> alter table basic\_ctrl\_net import tablespace;  Query OK, 0 rows affected, 1 warning (0.07 sec) |

5、查看数据库

|  |
| --- |
|  |

1. mysql中dblink的实现

如果数据库位于同一个mysql实例，可以通过在 database\_name.table\_name的形式来访问不同的数据库数据。当数据库位于不同的mysql实例上的时候，应该如何方位？

在oracle中可以通过建立数据链来建立远程连接，在mysql中有没有类似与oracle的dblink相似的功能呢？通过在网上查找，发现可以通过MySQL中的federated插件来实现类似的功能。

宿主机：127.0.0.1：3307 mysql 5.7

从机：127.0.0.1：3306 mysql 5.7

实现功能：

可以3306中MySQL数据库（target端）中建立宿主机3307的MySQL数据库（source端）中某个表的link，当在Linux中读取link表时，就相当于直接读取宿主机中的原始表内容。

9.1查看从机是否安装了federated插件：

|  |
| --- |
| 显示没有安装federated插件 |

9.2 安装federated插件

|  |
| --- |
| install plugin federated soname 'ha\_federated.so'; |
| 说明已经安装过了，但没有启用 |

9.3配置/etc/my.conf，设置federated为默认启动

在文件中加入一行：federated

重启mysql即可。

9.4在3307中建立表

|  |
| --- |
| use hkll;  CREATE TABLE `sl\_project` (  `PROJECT\_ID` varchar(32) NOT NULL COMMENT 'UUID,唯一标识',  `USER\_ID` varchar(32) DEFAULT NULL COMMENT '创建用户ID',  `PROJECT\_NAME` varchar(32) DEFAULT NULL COMMENT '模拟项目名称',  `CREATE\_TIME` timestamp NULL DEFAULT NULL COMMENT '创建时间',  `PROJECT\_TYPE` varchar(8) DEFAULT '2' COMMENT '1 示例工程 2 用户工程,示例工程一般用户不可删除',  `REMARK` varchar(200) DEFAULT NULL COMMENT '备注',  `NET\_ID` varchar(32) DEFAULT NULL COMMENT '电网模型ID',  `PERIOD\_NUM` int(11) DEFAULT NULL,  PRIMARY KEY (`PROJECT\_ID`),  KEY `FK\_Reference\_10` (`USER\_ID`),  KEY `sl\_project\_sl\_base\_net\_FK` (`NET\_ID`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='模拟的项目表，对应最外围的模拟项目'; |

9.5在3307中建立一个用户

|  |
| --- |
| GRANT SELECT ON \*.\* TO TEST@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;  备注：在此处只给TEST用户授予了SELECT权限，因此只能在从机中查询，不能做修改 |

9.6 在3306中建表

|  |
| --- |
| use test;  CREATE TABLE `sl\_project\_link77` (  `PROJECT\_ID` varchar(32) NOT NULL COMMENT 'UUID,唯一标识',  `USER\_ID` varchar(32) DEFAULT NULL COMMENT '创建用户ID',  `PROJECT\_NAME` varchar(32) DEFAULT NULL COMMENT '模拟项目名称',  `CREATE\_TIME` timestamp NULL DEFAULT NULL COMMENT '创建时间',  `PROJECT\_TYPE` varchar(8) DEFAULT '2' COMMENT '1 示例工程 2 用户工程,示例工程一般用户不可删除',  `REMARK` varchar(200) DEFAULT NULL COMMENT '备注',  `NET\_ID` varchar(32) DEFAULT NULL COMMENT '电网模型ID',  `PERIOD\_NUM` int(11) DEFAULT NULL,  PRIMARY KEY (`PROJECT\_ID`),  KEY `FK\_Reference\_10` (`USER\_ID`),  KEY `sl\_project\_sl\_base\_net\_FK` (`NET\_ID`)  ) ENGINE=FEDERATED  DEFAULT CHARSET=utf8 COMMENT='模拟的项目表，对应最外围的模拟项目'  CONNECTION='mysql://TEST:123456@127.0.0.1:3307/hkll/sl\_project' |

9.7在3306中查询数据

|  |
| --- |
|  |
| 如果修改sl\_project\_link77中的数据，会报错。因为TEST用户不具备修改数据权限 |